# An Incompleteness Theorem via Abstraction*

Alan Bundy[1], Fausto Giunchiglia[2,3], Adolfo Villafiorita[4,5] and Toby Walsh[2,5]

1. Mathematical Reasoning Group, Dept of AI, University of Edinburgh
2. Mechanized Reasoning Group, IRST
3. DISA, University of Trento
4. Istituto di Informatica, University of Ancona
5. Mechanized Reasoning Group, DIST, University of Genoa

April 13, 1996

**Abstract**

We demonstrate the use of abstraction in aiding the construction of an interesting and difficult example in a proof checking system. This experiment demonstrates that abstraction can make proofs easier to comprehend and to verify mechanically. To support such proof checking, we have developed a formal theory of abstraction and added facilities for using abstraction to the GETFOL proof checking system.

## 1 Introduction

This paper describes an experiment in which we use abstraction to aid the construction of a simplified proof of Gödel's first incompleteness theorem. We show that this use of abstraction makes the proof more accessible to both computer verification and human comprehension. This experiment also serves to illustrate some of the facilities implemented in the GETFOL proof checking system [Giu94].

Intuitively, an abstraction can be viewed as a mapping from a representation of a problem onto a new representation [GW92a]. We represent problems as goals to be proved inside appropriate first order theories. Let $<\Lambda, \Omega, \Delta>$ be an axiomatic formal system, where $\Lambda$ is a first order language, $\Omega$ is a set of axioms, and $\Delta$ is a set of inference rules. Then abstraction can be defined as a function $f$ which maps the language of a formal system onto the language of another formal system. The first formal system is called the *ground* representation (space/theory), while the second is called the *abstract* representation (space/theory). We also talk of abstract or ground language, axioms, and goal with the obvious meaning.

In [GW92a], we identify various forms of abstractions and show how they can be used in different ways. In this paper we consider an abstraction where the abstract axioms and

*The order is alphabetical and does not reflect the size of the contribution.

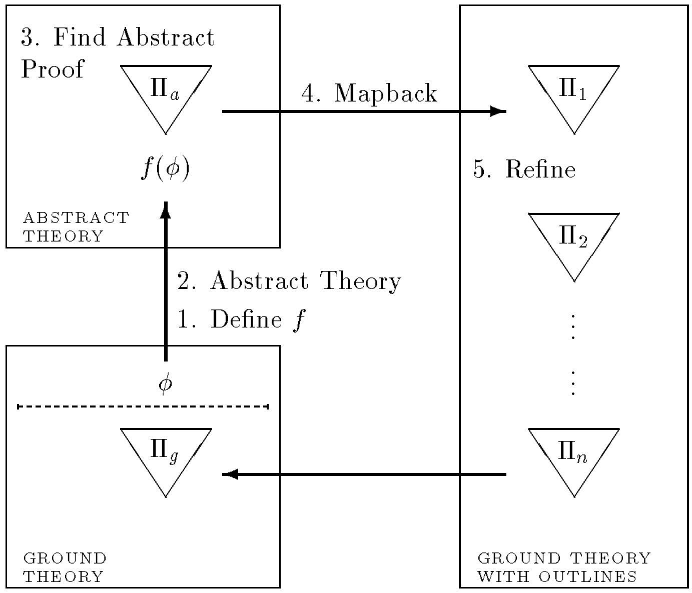


Figure 1: The cycle of theorem proving by abstraction.

goal are obtained by applying $f$ to the ground axioms and goal, and the abstract inference rules are the same as the ground ones. Furthermore, we restrict ourselves to a specific use of abstraction where the proof of the abstract goal is used to drive the construction of the proof of the ground goal. This can be thought of as a five-step process (see Figure 1). In the first step we define $f$. In the second step we apply $f$ to the ground representation to obtain the abstract representation. In the third step we prove the abstract goal, that is, we find a proof $\Pi_a$ of $f(\phi)$, where $\phi$ is the ground goal. $\Pi_a$ provides the key steps of the ground proof. In the fourth step we unabstract or *map back* $\Pi_a$, that is, we generate from $\Pi_a$ a tree $\Pi_1$, called *outline*, of schematic formulas [GW92b]. Schematic formulas are needed as usually $f$ is many-to-one; this allows us to build simpler and easier to solve abstract problems [GW92a]. The parameters occurring in the formulas in $\Pi_1$ represent possible choices in unabstracting abstract formulas into ground formulas. In the fifth and last step, called *refinement*, we refine $\Pi_1$ into a ground proof. This is achieved by building a sequence of outlines $\Pi_2, \ldots, \Pi_n$, where $\Pi_i$ (with $2 \leq i \leq n$) either instantiates a parameter of $\Pi_{i-1}$ or adds a proof step to $\Pi_{i-1}$; and then by checking that $\Pi_n$ actually represents a proof $\Pi_g$ of $\phi$. More details of this process are given in [GW92b] and also in [SVG94, VS94], where other examples of abstractions are presented.

GETFOL [Giu94], the proof checker used in this experiment, is an extension and re-implementation of the FOL proof checking system [Wey80]. GETFOL includes an extension of the Natural Deduction calculus [Pra65] (ND henceforth), syntactical and semantical simplification, and complex deciders for subclasses of first order logic. A single proof step in GETFOL can thus represent very complex reasoning. GETFOL also provides facilities for multi-theoretic reasoning. That is, GETFOL allows for multiple distinct logical theories (or, to use GETFOL terminology, *contexts*); this feature is essential for this experiment as the ground and the abstract representation are two distinct theories. We could perhaps call GETFOL an "interactive theorem prover". However we stick to "proof checker" as we wish to emphasize our interest in the interaction with the system rather than in the automation of the construction of proofs. Above all, GETFOL is a *conversational* system: the interface with GETFOL is designed to make the interaction with the user more a dialogue than a sequence of commands. The user engages in a conversation with GETFOL in which he or she describes the abstraction, builds an outline of the proof using this abstraction, and then progressively refines this outline. Potentially combinatorial explosive decisions are left to the judgment of the user (*eg.* choices in unabstracting formulas) while the system performs all the mechanical steps and book-keeping. The advantage over conventional approaches is that the proofs built are easier to understand and to construct. The two way nature of this conversation is, we believe, essential for abstraction to be useful.

In this paper we do not discuss more general issues about abstraction. We only notice that GETFOL is the first system where abstraction can be used interactively, combining it with user guidance, and inside a proof checking system. A comprehensive and general discussion about abstraction is given in [GW92a] (but see also [Pla81, GW89] for the use of abstraction in theorem proving, and [GW91, Kor87, Kno94] for some theoretical evidence of the advantages of using abstraction automatically). [GW92a] provides a long list of references to much work in this area, and rationally reconstructs some of the most important instances.

# 2 The Plan

The next Section briefly introduces Gödel's first incompleteness theorem. We present the original statement of the theorem, together with some basic definitions, and then translate a simplified axiomatization into the logic of the GETFOL system. Section 4 sketches a high level proof of the theorem in GETFOL. The actual proof by abstraction is presented in Sections 5, 6, and 7. Section 5 describes the first two steps of theorem proving by abstraction, that is, the declaration of the abstraction, and the generation of the abstract space. Section 6 describes the third step, namely the construction of the abstract proof. We introduce some of GETFOL's tools for building proofs, and explain the structure of the abstract proof. Section 7 describes the last two steps of theorem proving by abstraction, namely mapping back and refinement. Inside Section 7, Table 2 (page 20) compares the abstract proof with the ground proof. Finally, the Appendices collect some input/output of GETFOL: the ground language, the ground axiomatization, the abstract axioms, the abstract proof, and the ground proof can be found here.

GETFOL input and output are written using the `teletype` font. The text has been slightly edited, to make it more readable. Other formulas are written in *mathematical* font.

# 3 The Ground Theory

GETFOL allows for the definition and use of multiple theories at the same time, each of which has its own name. There is, however, only one current context, that is, only one theory in which at a given moment we can operate (e.g. define axioms, apply inference rules, and so on). When the system is started there is only one theory, which is also the current context. This theory has name `NOTNAMED&`, which actually means that it is unnamed. We begin therefore by giving this theory an appropriate name:

```
NOTNAMED&:: namecontext metamaths;
You have named the current context: metamaths
metamaths::
```

The `GETFOL` prompt is the name of the context inside which we are at the moment, followed by "`::`". Each `GETFOL` command has three parts: a string which uniquely identifies it, a list of arguments, and, finally, "`;`" which ends the command itself. `GETFOL` always produces a short answer which describes the action taken. With the above command we have given name `metamaths` to the context inside which we are going to carry out the ground proof.

Our goal is to prove Gödel First Incompleteness Theorem, which states that

> *"I. The system $S$ is not complete; that is, it contains propositions $A$ [...], for which neither $A$ nor $\neg A$ is provable [...] III. Theorem I can be sharpened to the effect that, even if we add finitely many axioms to the system $S$ [...] we do not obtain a complete system, provided the extended system is $\omega$-consistent." [Göd86]*

where

> *"a system is said to be $\omega$-consistent if, for no property $F(x)$ of natural numbers,*
>
> $$F(1),\ F(2),\ \ldots,\ F(n),\ \ldots \textit{ ad infinitum}$$
>
> *as well as*
>
> $$\exists\, x . \neg F(x)$$
>
> *are provable"* [Göd86].

In the original statement the system "S" is the logic of "Principia Mathematica" [WR25] with the axiom of choice (for all types) and the natural numbers as individuals. The theorem, however, can be proved for any formal theory containing arithmetic [Smo77].

We shall prove in `GETFOL` the following formula[1]:

$$OCONS \supset \exists\, w\ (\neg PROVABLE(w) \wedge \neg PROVABLE(\sim w)) \tag{1}$$

where $OCONS$ means that *maths* — a fixed but unspecified theory containing arithmetic — is $\omega$-consistent, $w$ is a variable ranging over formulas, $PROVABLE(w)$ means that the formula $w$ is provable in *maths*, and $\sim$ is the symbol for negation in *maths*. Notice that *maths* is not directly axiomatized inside `GETFOL`. Formula (1) is proven inside `metamaths`, a theory in which we discuss the provability, consistency, $\omega$-consistency, *etc.* of *maths*. `metamaths` is a formal metatheory of *maths* which formalizes (part of) the informal metatheory inside which Gödel carried out the proof of the incompleteness theorem.

[1]We write formulas which are not mechanized following the syntax presented in §17 of [Kle52] (with the following two exceptions: we use $\wedge$ instead of & and $p \leftrightarrow q$ as an abbreviation of $(p \supset q)\ \&\ (q \supset p)$).

| | |
|---|---|
| `w1, w2` | variables of `metamaths` ranging over formulas of *maths*; |
| `v1` | a variable of `metamaths` ranging over variables of *maths*; |
| `t1` | a variable of `metamaths` ranging over terms of *maths*; |
| `n` | a variable of `metamaths` ranging over natural numbers; |
| `x` | a constant of `metamaths` representing a variable of *maths*; |
| `PROVABLE(w1)` | the formula `w1` is provable in *maths*; |
| `CONS` | *maths* is consistent; |
| `OCONS` | *maths* is $\omega$-consistent; |
| `SUBST(t1,v1,w1,w2)` | substituting the term `t1` for the variable `v1` in the formula `w1` gives the formula `w2`; |
| `NUMBER(n)` | `n` is a natural number; |
| `prov(w1, n)` | `n` is the Gödel number of a proof of the formula `w1`; |
| `~ w1` | `~` is the symbol for negation in *maths*; |
| `w1 equiv w2` | `equiv` is the symbol for logical equivalence in *maths*; |
| `all(x, w1)` | `all` is the symbol for universal quantification in *maths*. |

Table 1: Intended interpretation of some symbols of the alphabet of `metamaths`.

As such, `metamaths` has terms which denote elements of *maths* and variables ranging over such elements. Consider for instance formula (1). We said above that $w$ ranges over formulas and that $\sim$ is the symbol for negation in *maths*. To be precise, we should have said that $w$ ranges over terms denoting formulas of *maths*, and that $\sim$ is a function symbol such that the term obtained by applying it to a term denoting a formula of *maths* denotes the negation of the formula itself. To keep explanations simple, from now on we leave implicit all of this and speak freely, for instance, of variables ranging over elements of *maths*. We will be precise only when this is necessary for a correct understanding of what is going on.

By default, each `GETFOL` context has a first order language. The user is left with the task of defining its alphabet. This is done using the `declare` command. Consider for instance the following command:

```
metamaths:: declare predconst PROVABLE 1;
PROVABLE has been declared to be a Predconst
```

This command declares `PROVABLE` as a predicate of arity 1. The complete definition of the alphabet is given in the Appendix. The intended interpretation of some important symbols is given in Table 1.

We next give some axioms from which we will prove formula (1). These axioms define various important properties of *maths*. First we state what it means for *maths* to be consistent[2]:

[2] Mechanized formulas follow the syntax of the `GETFOL` system ([Giu94], Section 7). In particular, quantifiers may have more than one variable; a "`.`" is required after the last variable in the scope of a quantifier. Thus, for instance, `forall x y.p` stands for $\forall x\,(\forall y\,p)$.

```
cons: forall w1.(CONS imp not PROVABLE(w1) or not PROVABLE(~ w1))
```

GETFOL axioms have names. In a GETFOL axiom the string before "`:`" — in this case `cons` — is the name used to refer to the axiom.

We also state as axioms two consequences of $\omega$-consistency:

```
occ: OCONS imp CONS;
oc: OCONS imp not (forall n. (NUMBER(n) imp PROVABLE(diag(n))) and
                  PROVABLE(~ all(x, diag(x))))
```

Axiom `occ` states that $\omega$-consistency is stronger than consistency (see [Kle52] for a proof). Axiom `oc` is an instance of the definition of $\omega$-consistency (see below for an explanation of the meaning of `diag`).

In `metamaths` we also want a formula of the type:

$$\forall w\ (PROVABLE(w) \leftrightarrow \exists\, n\ (NUMBER(n) \wedge\ PROVABLE(prov(fno(w), n))))$$

where $prov(fno(w), n)$ is true when $n$ is the Gödel number of a proof of the formula whose Gödel number is $fno(w)$. To simplify the proof and avoid the manipulation of the existential quantifier, we replace the above formula with the following two implications

```
corr: forall w1. (PROVABLE(w1) imp
                 PROVABLE(prov(w1, k(w1))) and NUMBER(k(w1)))
comp: forall w2 n. (NUMBER(n) and PROVABLE(prov(w2, n)) imp PROVABLE(w2))
```

where `k` is a Skolem function introduced to eliminate the existential quantifier; `k` takes as argument a formula and returns a number. Notice that the conjunction of `corr` and `comp` is not equivalent to the formula above. Note also that `prov`, differently from *prov*, takes as first argument a formula rather than a number. Thus `prov(w,n)` in the mechanized proof stands for $prov(fno(w), n)$.

Consider now the following formula, built using Cantor's diagonal method (see [Kle52], page 207 for more details):

$$all(x, \sim prov(p, x)) \tag{2}$$

where $p$ is the Gödel number of the formula $all(x, \sim prov(p, x))$. Thus (2) is a formula which asserts its own unprovability. Notice that $all(x, \sim prov(p, x))$ is such that neither (the formula denoted by) $all(x, \sim prov(p, x))$ itself nor its negation is provable in *maths*. In the GETFOL proof we start from the following formula:

$$PROVABLE(all(x, \sim prov(p, x)\ equiv\ \sim prov(fno(all(x, \sim prov(p, x))), x))) \tag{3}$$

In order to enhance readability, we use $diag(x)$ as a synonym of $\sim prov(p, x)$. We have therefore the following axiom:

```
diagonal: PROVABLE(all(x, diag(x) equiv (~ prov(all(x, diag(x)), x))))
```

The formula `all(x,diag(x))` and its negation are the formulae that we will show unprovable in *maths*. Note that, since `~prov(all(x,diag(x)),x)` means that `x` is not the Gödel number of a proof of `all(x,diag(x))`, the formula `all(x,diag(x))`, which is a synonym of `all(x,~prov(all(x,diag(x)),x))`, expresses its own unprovability.

An important property of `diag` is that it is numeralwise expressible [Kle52]:

```
numwise: forall n. (NUMBER(n) imp PROVABLE(diag(n)) or PROVABLE(~ diag(n)))
```

That is, for any natural number `n`, either `diag(n)` or `~diag(n)` is provable in *maths*. We also need to describe some of the inference rules of *maths*:

```
iffel:  forall w1 w2. (PROVABLE(w1) and PROVABLE(w1 equiv w2) imp
                       PROVABLE(w2))
iffers: forall w1 w2. (PROVABLE(~ w1) and PROVABLE(w1 equiv ~ w2) imp
                       PROVABLE(w2))
alle:   forall v1 w1 t1 w2. (PROVABLE(all(v1, w1)) and
                             SUBST(t1, v1, w1, w2) imp PROVABLE(w2))
```

The axiom `iffel` describes the $\leftrightarrow E_{left}$ rule of *maths*. The axiom `iffers` encodes the application of a $\leftrightarrow E_{right}$, a $\neg E$, and a $\perp_c$, shortening both the axiomatization and the proof. The axiom `alle` describes the $\forall E$ inference rule. Intuitively, `SUBST(t1,v1,w1,w2)` is true when the formula `w2` is the result of uniformly substituting the term `t1` for the variable `v1` in the formula `w1`. The axiomatization is completed by three axioms describing some instances of `SUBST`. For instance:

```
triv1: SUBST(k(all(x, diag(x))), x, diag(x), diag(k(all(x, diag(x)))))
```

says that the result of substituting `k(all(x,diag(x)))` for `x` in `diag(x)` is the formula `diag(k(all(x,diag(x))))`. Notice that, as expected, `k` takes as argument a formula (that is, `all(x, diag(x))`), and `diag` a number (that is, `k(all(x, diag(x)))`).

The complete set of axioms is given in the Appendix. As a conclusive remark, it is important to notice that the axioms of `metamaths` make the proof much simpler and shorter than it would be if we started from first principles, e.g. from the axioms of Peano Arithmetic. However this observation does not weaken the message of this paper, which shows how abstraction helps in that part of the proof where, starting from the `diagonal` axiom (and the other principles listed above), we prove that there is a formula $w$ such that neither $w$ nor its negation is provable. (See Section 8 for a longer discussion on this point.)

# 4 The Ground Proof

In this Section we will just discuss the key steps of the ground proof. The full proof is given in the Appendix.

The proof starts with an assumption, discharged at the very end, that *maths* is $\omega$-consistent:

```
11   OCONS     (11)
```

In `GETFOL`, proof steps, when asserted, are printed out as (proof) lines (also called *facts*). A line consists of three parts. In order these are: the line number, the formula derived at this point in the proof, and the set of its dependencies (that is, the set of the line numbers of the assumptions on which it depends). In this case, we are at the line labeled `11`, and have a formula `OCONS` whose set of dependencies is the singleton set with unique element `11`[3].

The proof now divides in two halves. In the first half we prove, by *reductio ad absurdum*, that `all(x,diag(x))` is unprovable. We therefore start by assuming that `all(x,diag(x))` is indeed provable:

```
12   PROVABLE(all(x, diag(x)))     (12)
```

By applying the `alle` axiom, from assumption `12`, substituting `k(all(x,diag(x)))` for `x` we get:

```
13   PROVABLE(diag(k(all(x, diag(x)))))     (12)
```

and, by using proof line `13`, the `diagonal` axiom (to get the provability in *maths* of `diag(k(all(x, diag(x)))) equiv ~ prov(all(x, diag(x)), k(all(x, diag(x))))`), and the `iffel` axiom, we get:

```
14   PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x)))))     (12)
```

On the other hand, by using `corr` and assumption `12` we derive that:

```
15   PROVABLE(all(x, diag(x))) imp
     PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x))))) and
     NUMBER(k(all(x, diag(x))))
16   PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))     (12)
```

We have thus found a contradiction, since — if *maths* is consistent (assumption `11` and axiom `occ`) — it cannot be the case that a formula and its negation are both provable (proof lines `14` and `16`):

```
10   CONS imp not PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x))))) or
              not PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))
17   FALSE     (11 12)
```

`FALSE` is the symbol for falsity ($\perp$) in the `GETFOL` language. Proof line `17` is derived from `10`, `14`, and `16` in just one step using `GETFOL`'s tautological decider `taut`. The input/output behavior of `taut` is described in [Giu94]; the details of its implementation are described in [AG93, GAP95]. Here it suffices to know that `taut` takes a goal formula (in this case the formula of line `17`) and a set of hypotheses (in this case `10`, `14`, and `16`) and tries to prove that the goal follows from the hypotheses by applying propositional reasoning only. If this is the case, then `taut` asserts the goal as a proof line with appropriate dependencies.

We can now discharge the initial assumption, by applying the $\perp_c$ rule to proof line `17`, thus concluding the first half of the proof:

[3] In this section, contrarily to what happens in the rest of the paper, proof steps are written without the commands which generates them. The reason is that the proof outlined in this section is generated by the commands described in Section 7. Notice furthermore that the label of the above proof line is `11` and not `1`, as one could have expected, and as it would have been if we had proved the goal using the inference rules of the ground space. The labeling given in this section is that generated via abstraction, and is described in Section 7.

```
18   not PROVABLE(all(x, diag(x)))     (11)
```

In the second half of the proof we show that `~all(x, diag(x))` is unprovable. The proof is again by *reductio ad absurdum*. We start by assuming

```
19   PROVABLE(~ all(x, diag(x)))     (19)
```

From this assumption, the assumption of $\omega$-consistency (proof line `11`), and axiom `oc` we can prove

$$\neg \forall n \, (NUMBER(n) \supset PROVABLE(diag(n)))$$

which is in turn equivalent to

```
20   exists n. (not (NUMBER(n) imp PROVABLE(diag(n))))     (11 19)
```

In the `GETFOL` proof, proof line `20` is proved using `GETFOL`'s command `monad`. As with `taut`, `monad`'s input/output behavior is described in [Giu94], while the details of its implementation are in [AG93, GAP95]. Here it suffices to know that `monad` decides a class which contains the monadic class, the $\forall\exists$ class and a class which reduces to the $\forall\exists$ class by simple quantifier manipulation.

We can now apply `GETFOL`'s $\exists$E rule. This requires us to make the following assumption:

```
21   not (NUMBER(n) imp PROVABLE(diag(n)))     (21)
```

(`GETFOL`'s $\exists$E rule actually implements existential instantiation [Giu94].) From `21`, since `diag` is numeralwise expressible, we have

```
22   PROVABLE(~ diag(n))     (21)
```

Using the `diagonal` and the `iffers` axioms, we get:

```
23   PROVABLE(prov(all(x, diag(x)), n))     (21)
```

The proof is now almost done, since, by completeness

```
24   (NUMBER(n) and PROVABLE(prov(all(x, diag(x)), n))) imp
     PROVABLE(all(x, diag(x)))
25   PROVABLE(all(x, diag(x)))     (11 19)
```

which is clearly in contradiction with proof line `19` and the assumption of $\omega$-consistency of *maths*:

```
26   CONS imp (not PROVABLE(all(x, diag(x))) or
               not PROVABLE(~ all(x, diag(x))))
27   FALSE     (11 19)
```

By applying the $\perp_c$ rule we get

```
28   not PROVABLE(~ all(x, diag(x)))     (11)
```

The last three lines of the proof build the goal formula. We first apply an $\wedge$I to the proof lines showing the unprovability of `all(x,diag(x))` and its negation; we then apply an $\exists$I to the formula thus obtained; we finally discharge the initial assumption of $\omega$-consistency with an $\supset$I:

```
29   (not PROVABLE(all(x, diag(x)))) and
     (not PROVABLE(~ all(x, diag(x))))     (11)
30   exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (11)
31   OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
```

Although this proof may have seemed quite long and tortuous (31 steps), it was considerably shortened by the use of GETFOL's decision procedures — the proof is more than 70 steps using just the introduction and elimination rules of GETFOL's ND logic. The proof itself consists of universal eliminations on the axioms, propositional reasoning using the `taut` decider, predicative reasoning using the `monad` decider, and a final existential introduction. As well as being complete for fragments of first order logic, GETFOL's deciders are also very efficient. For example, given the appropriate axioms and proof lines, `monad` is able to show in just one step (as opposed to 10 ND steps) and $\sim 0.3$ seconds run time (on a SPARC station 10[4] running GETFOL on GNU common LISP 1.0) that the formula of line 20, that is `exists n.(not (NUMBER(n) imp PROVABLE(diag(n))))`, is provable. Of course, the difficult problem is in determining which instances of the axioms to use and in choosing the formula to be proved.

This proof turns out to be a considerable challenge to an unguided theorem prover. We have given these axioms to OTTER (v. 3.0) [McC90] but it blew up. Using *"ordered hyper resolution, unit deletion, and factoring, with satellites in sos with and nuclei in usable"*, 30093733 clauses (of which 11632 were kept) were generated after 2 days on a Sun SPARC station 10. In the rest of the paper, we will show how abstraction can help tackle these problems.

# 5 The Abstract Theory

More than 10 years ago, the first author gave a slightly different version of the axioms described in Section 3 to a Prolog-based resolution theorem prover. Motivated by its failure to prove a variant of formula (1), he proposed (what we now call) an abstraction which simplifies the proof greatly, halving it in size. Following [GW92a], we now have the formal machinery necessary to mechanize this proposal in a general and provably correct way.

The abstraction we propose maps terms of `metamaths` onto new terms of the abstract theory. All the other aspects of the ground language (for example, the predicate symbols, the logical connectives) are left unchanged. Formally, this is an *atomic* abstraction [GW92a]. Such abstractions were first proposed by Plaisted [Pla81]. The goal is to make the `diagonal` axiom redundant and thus avoid reasoning (in the abstract space) about the details of diagonalization. More precisely, we map the predicates of *maths* (that is, terms of `metamaths`) onto sentential letters (that is, individual constants of the abstract theory) in such a way that the `diagonal` axiom becomes a tautology. This is obtained

[4] SUNW, SPARC station-10 microprocessor, 32M RAM, Sun OS 4.1.3 U1 operating system.

as follows: `diag(X)` and `~prov(X,Y)` are mapped onto `d` (this mapping transforms the diagonalization formula onto a tautology); `prov(X,Y)` is mapped onto `~d` (this mapping is coherent with the previous, which changed the polarity of `prov`); `all(X,Y)` is mapped onto the abstraction of `Y` (notice that since predicates of *maths* are mapped onto sentential letters, universal quantifications are useless in the abstract theory). It is important to notice that this abstraction is not meaning preserving, that is, different ground terms and formulas are collapsed into a single abstract term or formula. This is often the case when using abstraction [GW92a]. The hope is that the abstract goal will be much easier to solve. Of course, the existence of a proof of the abstract goal does not guarantee the existence of a proof of the ground goal. This must be checked, and it is the goal of the mapping back and refinement. In particular, in this case, in the ground space we have to fit into the abstract proof all the extra steps which take into account the details of diagonalization. The advantage is that these (often confusing) details are dealt with in isolation, when all the rest of the proof has been carried out.

The `GETFOL` definition of the abstraction is as follows:

```
metamaths:: makecontext absmetamaths;
metamaths:: abstraction declare goedel: metamaths => absmetamaths
   TABS(all($x, $y))        := TABS($y)
   TABS(diag($x))           := d
   TABS(~ prov($x, $y))     := d
   TABS(prov($x, $y))       := ~ d

   WABS($A and $B)          := WABS($A) and WABS($B)
   WABS(forall $x. $A)      := forall TABS($x). WABS($A)
   ...

   WABS(PROVABLE($x))       := PROVABLE(TABS($x))
   WABS(NUMBER($x))         := NUMBER(TABS($x))
   WABS(OCONS)              := OCONS
   ...
;
```

In `GETFOL`, abstractions have names. The name chosen for this abstraction is `goedel`. An abstraction is defined by giving a ground context, in this case `metamaths`, an abstract context, in this case `absmetamaths`, and the mapping $f$ between the languages of the two contexts. The mapping $f$ is defined as a set of rewrite rules: `TABS` (which stands for `T`erm `ABS`traction) defines the rewriting rules of $f$ over terms; `WABS` (which stands for `W`ell formed formula `ABS`traction) defines the rewriting rules of $f$ over formulas. In the definition of `goedel`, the symbols prefixed by a "`$`" sign are schematic variables which can be substituted with formulas or terms (depending on their names). Thus, for instance, `diag($x)` represents all the formulas of *maths* having `diag` as outermost predicative symbol, like `diag(x)`, or `diag(k(all(x,diag(x))))`. Notice that `TABS` and `WABS` are recursively defined over the structure of terms and formulas respectively. Thus, for instance, the rule `TABS(all($x, $y)) := TABS($y)` says that the abstraction of any universally quantified formula of *maths* is equal to the abstraction of the formula in the scope of the quantification. This rewriting rule removes universal quantifications from formulas of *maths*. Finally, the command `makecontext`, executed just before the definition of `goedel`, creates a new `GETFOL` context and gives it name `absmetamaths`.

Having defined the abstraction, we can now perform the second step of Figure 1 (page 2), that is, apply $f$ to obtain the abstract space. This is done by feeding the set of rewrite rules defining $f$ into the `GETFOL` rewriter. The resulting axioms, formulas and goals are then automatically asserted as such in the abstract space. The complete set of abstract axioms is reported in the Appendix.

To demonstrate the mapping of the language, let us consider for instance how the third line of the proof from the bottom maps into the abstract space:

```
metamaths:: abstract wff (not PROVABLE(all(x,diag(x))) and
                          not PROVABLE(~ all(x, diag(x)))) by goedel;
I am switching from the current context to: absmetamaths
(not PROVABLE(d)) and (not PROVABLE(~ d))
is the mapped wff from 'metamaths'.
```

The last line of the ground proof (which does not involve any of details of diagonalization) is instead mapped across without change.

The axioms are mapped across similarly. Consider the following example:

```
metamaths:: abstract axiom iffel by goedel;
I am switching from the current context to: absmetamaths
iffel : forall w1 w2. ((PROVABLE(w1) and PROVABLE(w1 equiv w2)) imp
                       PROVABLE(w2))
has been declared to be a new axiom in 'absmetamaths'.
```

As expected, this and the other axioms describing the inference rules of *maths* are not changed by the abstraction function. Consider now the mapping of the `diagonal` axiom:

```
metamaths:: abstract axiom diagonal by goedel;
I am switching from the current context to: absmetamaths
diagonal : PROVABLE(d equiv d)
has been declared to be a new axiom in 'absmetamaths'.
```

Again, as expected, the `diagonal` axiom maps onto the provability of a tautological formula, `d equiv d`. Not surprisingly, we do not need to use it in the abstract proof.

# 6 The Abstract Proof

The third step of Figure 1 consists of finding a proof in the abstract context. This is much easier than finding a proof in the ground context. Indeed, it is well within the reach of current resolution theorem provers. OTTER (v. 3.0), for example — using the setting described in Section 4 for the ground proof — was able to find a proof in just 0.14 seconds of user CPU time on a SPARC station 10, generating just 40 clauses (of which 34 were kept). This resolution proof is an outline of a ground proof. Similarly, as we demonstrate in the next section, the abstract ND proof given in this section serves as an outline for the ground ND proof given in Section 4.

To build the abstract proof we need to switch to `absmetamaths`, that is, to make `absmetamaths` the current context. This is achieved by the following `GETFOL` command:

```
metamaths:: switchcontext absmetamaths;
You are now using context: absmetamaths
absmetamaths::
```

As before, the proof begins with the assumption of $\omega$-consistency of *maths*,

```
absmetamaths:: assume OCONS;
1   OCONS     (1)
```

Like in the ground proof, the abstract proof divides naturally into two halves. In the first half we show — by *reductio ad absurdum* — that `d` is unprovable. We begin therefore by assuming the opposite of the goal

```
absmetamaths:: assume PROVABLE(d);
2   PROVABLE(d)     (2)
```

The next four lines of the proof are devoted to deriving from this a contradiction. The contradiction is easier to see than in the ground axiomatization. By axiom `corr`, if `d` is provable then `~d` will also be provable. We apply $\forall$E to axiom `corr` and the `taut` command to proof lines 2 and 3:

```
absmetamaths:: alle corr d;
3   PROVABLE(d) imp (PROVABLE(~ d) and NUMBER(k(d)))

absmetamaths:: taut PROVABLE(~ d) by 2 3;
4   PROVABLE(~ d)     (2)
```

`GETFOL`'s commands for ND's inference rules are composed of a string identifying a logical symbol (*eg.* `all` for universal quantification) suffixed by `e` or `i`, for the elimination or the introduction rule respectively.

If *maths* is consistent (assumption `1` and axiom `occ`), a formula and its negation cannot both be provable. Thus we have found the contradiction we seek and we can discharge assumption `2` by asserting its negation. Let us consider this argument in detail. By the definition of consistency we have:

```
absmetamaths:: alle cons d;
5   CONS imp ((not PROVABLE(d)) or (not PROVABLE(~ d)))
```

Proof line `5` is in contradiction with line `1`, axiom `occ`, line `2`, and line `4`:

```
absmetamaths:: taut FALSE by 1 occ 5 2 4;
6   FALSE     (1 2)
```

Thus we can discharge the assumption that `d` is provable:

```
absmetamaths:: noti 6 2;
7   not PROVABLE(d)     (1)
```

In the second half of the proof we show that `~d` is also unprovable. As in the ground proof, we show that if `~d` were provable, `d` would also be provable, which is in contradiction with the assumption of ($\omega$-)consistency of *maths*. We start therefore by assuming the provability of the formula `~d`

```
absmetamaths:: assume PROVABLE(~ d);
8   PROVABLE(~ d)     (8)
```

But, by axiom `oc`, the previous proof line, and assumption `1`, using the decider `monad`:

```
absmetamaths:: monad exists n. not (NUMBER(n) imp PROVABLE(d)) by 8 1 oc;
9   exists n. (not (NUMBER(n) imp PROVABLE(d)))     (1 8)
```

We now eliminate the existential quantifier and show, using `comp`, proof line `8` and proof line `10`, that `d` is provable:

```
absmetamaths:: existe 9 n;
10   not (NUMBER(n) imp PROVABLE(d))     (10)

absmetamaths:: alle comp d n;
11   (NUMBER(n) and PROVABLE(~ d)) imp PROVABLE(d)

absmetamaths:: taut PROVABLE(d) by 8 10 11;
12   PROVABLE(d)     (1 8)
```

Proof lines `8` and `12` are in contradiction with the consistency of *maths*:

```
absmetamaths:: alle cons d;
13   CONS imp ((not PROVABLE(d)) or (not PROVABLE(~ d)))
```

We can discharge assumption `8` asserting its negation:

```
absmetamaths:: taut FALSE by 1 occ 1 8 12 13;
14   FALSE     (1 8)

absmetamaths:: noti 14 8;
15   not PROVABLE(~ d)     (1)
```

The proof is now almost done. The goal is built by introducing a conjunction, an existential, and by finally discharging the initial assumption of $\omega$-consistency with the introduction of an implication:

```
absmetamaths:: andi 7 15;
16   (not PROVABLE(d)) and (not PROVABLE(~ d))     (1)

absmetamaths:: existi 16 d:w1;
17   exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (1)

absmetamaths:: impi 1 17;
18   OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
```

The proof requires just 5 of the abstract axioms. Indeed, OTTER was able to find an abstract proof slightly quicker when given just these 5 axioms as opposed to all of them (0.09 seconds on a SPARC station 10 generating 26 clauses of which 16 were kept using the same settings as the other examples). Note also that the abstract proof does not require any of the complicated axioms, *eg.* `diagonal` or `numwise`.

A considerable use of the deciders was made in the proof; they considerably reduced the length of the proof. Given the appropriate premisses, the decision procedure `taut` is able to determine in just one step the formula of line 12, that is `PROVABLE(d)`. The fact that this takes 0.20 seconds run time on a SPARC station 10 and that the same command in the ground theory fails (the corresponding ground formula is too complex and refinement is needed to simplify the inference) is an indication of the reduction in complexity abstraction had provided here. Of course, as in the ground space, the difficult problem is in determining which instances of the axioms to use and in choosing the formula to be proved.

# 7 Mapping Back and Refinement

Steps four and five of Figure 1 are mapping back and refinement, that is, the transformation of the abstract proof into a ground proof. From the abstract proof we build an outline of the ground proof. This outline contains parameters and deductions corresponding to the key steps of the ground proof. We must therefore refine this outline. That is, the user must guide the system in the choice of instantiating parameters and of adding missing steps. To perform such refinement steps in a proof checking system, we need some commands for manipulating outlines. In addition, we need to distinguish between the steps in an outline and those in a proof: steps in an outline may not be derivable as mapping back is not guaranteed. Steps in an outline are merely conjectures. We call them, *tries.*

For simplicity, we begin with the conclusion of the abstract proof

```
absmetamaths:: show proof;
...
16   (not PROVABLE(d)) and (not PROVABLE(~ d))     (1)
17   exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (1)
18   OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
```

`show proof` shows all the steps of the proof being built, following the order in which they have been asserted. We first instruct `GETFOL` to map the last three facts back to tries in the ground space.

```
absmetamaths:: mapback fact 16 by goedel;
absmetamaths:: mapback fact 17 by goedel;
absmetamaths:: mapback fact 18 by goedel;
absmetamaths:: switchcontext metamaths;
You are now using context: metamaths
metamaths:: show outline;
16.0  (not PROVABLE(d)) and (not PROVABLE(~ d))
       ANDI  7.0  15.0
17.0  exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
       EXISTI  16.0
18.0  OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
       IMPI  OCONS  17.0
```

`show outline` shows all the steps of the outline being built. A try has a number, a formula, and a justification, which explains how the try has been asserted. In this case, the tries are numbered `16.0`, `17.0`, and `18.0`; and, for instance, try `18.0` has been obtained with an $\supset$I between the formula `OCONS` and the try `17.0`. To add extra steps between these tries, `GETFOL` would use line numbers of the form `16.`$n$, `17.`$n$, and `18.`$n$, with increasing $n$. Note that `17.0` and `18.0` are the unique unabstractions of lines `17` and `18` in the abstract proof. Line `16`, by comparison, has several possible unabstractions, all due to possible instantiations of `d` and `~ d`. The formula displayed for try `16.0` is meant to represent one of a set of ground formulas which abstract onto line `16`. Notice indeed that `d` is not an element of the language of `metamaths` (see Appendix); it is rather a parameter which must be substituted by one of the formulas that the abstraction `goedel` maps onto it. We call this process, (parameter) *instantiation*. `GETFOL` guarantees the correctness of instantiations. The legal instantiations for try `16.0` are only those yielding a ground formula that abstracts to proof line `16` of the abstract proof. In this case, we decide to instantiate all occurrences of `d` in the try `16.0` to the term `all(x,diag(x))` with the command `tryinst` (**inst**antiate a **try**):

```
metamaths:: tryinst 16.0 d: all(x, diag(x)) all by goedel;
16.0  (not PROVABLE(all(x, diag(x)))) and (not PROVABLE(~ all(x, diag(x))))
       ANDI  7.0  15.0
```

Try `17.0` now follows immediately from `16.0`. To show this we perform an existential introduction on `16.0`, and match the result of this existential introduction with `17.0`.

```
metamaths:: tryexisti 16.0 all(x, diag(x)):w1;
16.1  exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))  17.0
       EXISTI  16.0

metamaths:: trymatch 16.1 17.0;
17.0 has been bound to 16.1.
```

All the commands manipulating tries start with `try`; in particular all the commands implementing ND inference rules which apply to tries have their usual name prefixed by `try`. The command `trymatch` takes two tries and verifies that they are actually two distinct instances of the same try.

So far, we have successfully mapped back and refined the penultimate two lines of the proof. However, rather than laboriously mapping back each line of the abstract proof individually, we can map back the whole of the abstract proof using one command:

```
metamaths:: switchcontext absmetamaths;
You are now using context: absmetamaths
absmetamaths:: mapback proof by goedel;
```

This creates an outline of 18 lines, each being a parameterized unabstraction of the corresponding line in the abstract proof. Let's switch now our attention to the middle of the proof. In the abstract proof, at line `8`, we assumed `PROVABLE(~ d)`. The try associated with this line is thus:

```
absmetamaths:: switchcontext metamaths;
You are now using context: metamaths
```

```
metamaths:: show outline;
...
8.0  PROVABLE(~ d)
       ASSUME  PROVABLE(~ d)
...
```

The refinement of try `8.0` consists of choosing an instantiation for `d`. This is easy since in refining try `16.0`, we committed to `all(x,diag(x))` as the unabstraction of `d`, the unprovable formula. We therefore instantiate `d` to `all(x,diag(x))` in try `8.0`:

```
metamaths:: tryinst 8.0 d: all(x, diag(x)) by goedel;
8.0  PROVABLE(~ all(x, diag(x)))
       ASSUME  PROVABLE(~ all(x, diag(x)))
```

This demonstrates an important feature of refinement. `GETFOL` is a system traditionally used to reason forwards from axioms and hypotheses to a goal. Facilities have also been added for reasoning backwards from the goal to the axioms and hypotheses. In reasoning with outlines, however, the user has greater flexibility; they can reason from the beginning of the proof, from the end, or (if they wish) from the middle.

We must now find an instantiation for tries `9.0`, `10.0`, `11.0`, and `12.0`:

```
metamaths:: show outline;
...
9.0  exists n. (not (NUMBER(n) imp PROVABLE(d)))  10.0
       MONAD  exists n. (not (NUMBER(n) imp PROVABLE(d))) BY 8.0 1.0 oc
10.0  not (NUMBER(n) imp PROVABLE(d))
       EXISTE  9.0
11.0  (NUMBER(n) and PROVABLE(~ d)) imp PROVABLE(d)
       ALLE  comp  d, n
12.0  PROVABLE(d)
       TAUT  PROVABLE(d) BY 8.0 10.0 11.0
...
```

By looking at the justifications of try `11.0`, we decide to instantiate the parameter `~ d` to `prov(all(x,diag(x)),n)` and the parameter `d` to `all(x,diag(x))`. Try `11.0` is now derivable by applying `ALLE` to axiom `comp`.

```
metamaths:: tryinst 11.0 ~ d: prov(all(x,diag(x)),n) all by goedel;
11.0  (NUMBER(n) and PROVABLE(prov(all(x, diag(x)), n))) imp
      PROVABLE(d)
       ALLE  comp  d, n
metamaths:: tryinst 11.0 d: all(x,diag(x)) all by goedel;
11.0  (NUMBER(n) and PROVABLE(prov(all(x, diag(x)), n))) imp
      PROVABLE(all(x, diag(x)))
       ALLE  comp  all(x, diag(x)), n
```

Reasoning in a similar way, we decide to instantiate the parameter `d` in tries `9.0` and `10.0` to `diag(n)`:

```
metamaths:: tryinst 9.0 10.0 d: diag(n) by goedel;
9.0  exists n. (not (NUMBER(n) imp PROVABLE(diag(n))))
       MONAD  exists n. (not (NUMBER(n) imp PROVABLE(diag(n)))) BY 8.0 1.0 oc
10.0  not (NUMBER(n) imp PROVABLE(diag(n)))
       EXISTE  9.0
```

Try `9.0` can now be obtained by applying `monad` to try `8.0`, `1.0`, and axiom `oc`; try `10.0` by applying `EXISTE` to try `9.0`. As for the try `12.0`, we instantiate the parameter `d` to `all(x,diag(x))`, so that from try `12.0` we derive a contradiction with try `8.0` and axiom `cons`:

```
metamaths:: tryinst 12.0 d: all(x,diag(x)) all by goedel;
12.0  PROVABLE(all(x, diag(x)))  nobelow
       TAUT  PROVABLE(all(x, diag(x))) BY 8.0 10.0 11.0
```

There is now a gap between try `12.0` and its premisses. That is, try `12.0` is not obtainable by applying the `taut` decider: we need to add a few steps to the outline. When this happens the gap must always be filled by using one of facts made useless by the abstraction. In this case we use the `numwise` axiom.

```
metamaths:: tryalle numwise n;
0.5  NUMBER(n) imp (PROVABLE(diag(n)) or PROVABLE(~ diag(n)))
       ALLE  numwise  n

metamaths:: trytaut PROVABLE(~ diag(n)) by 0.5 10.0;
10.1  PROVABLE(~ diag(n))
       TAUT  PROVABLE(~ diag(n)) BY 0.5 10.0
```

The next two steps are devoted to proving `PROVABLE(diag(n) equiv ~ prov(all(x, diag(x)), n)`.

Using the `diagonal` and the `alle` axioms:

```
metamaths:: tryalle alle
                    x diag(x) equiv ~ prov(all(x, diag(x)), x)
                    n diag(n) equiv ~ prov(all(x, diag(x)), n);
0.7  (PROVABLE(all(x, diag(x) equiv (~ prov(all(x, diag(x)), x)))) and
     SUBST(n, x, diag(x) equiv (~ prov(all(x, diag(x)), x)),
                diag(n) equiv (~ prov(all(x, diag(x)), n)))) imp
     PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))
       ALLE  alle ...

metamaths:: trytaut PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))
            by diag 0.7 triv6;
0.8  PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))
       TAUT  PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n))) ...
```

By using the `iffers` axiom, try `0.8`, and try `10.1`:

```
metamaths:: tryalle iffers diag(n) prov(all(x,diag(x)),n);
0.6  (PROVABLE(~ diag(n)) and
     PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))) imp
```

```
    PROVABLE(prov(all(x, diag(x)), n))
      ALLE  iffers  diag(n), prov(all(x, diag(x)), n)

metamaths:: trytaut PROVABLE(prov(all(x, diag(x)), n)) by 10.1 0.6 diagonal;
10.2  PROVABLE(prov(all(x, diag(x)), n))
      TAUT  PROVABLE(prov(all(x, diag(x)), n)) BY 10.1 0.6 0.8
```

We are now almost done. By axiom `comp` (try `11.0`), try `10.2`, and try `10.0` (from which we know `NUMBER(n)`) we get:

```
metamaths:: trytaut PROVABLE(all(x, diag(x))) by 10.2 10.0 11.0;
11.1  PROVABLE(all(x, diag(x)))
      TAUT  PROVABLE(all(x, diag(x))) BY 10.2 10.0 11.0

metamaths:: trymatch 12.0 11.1;
12.0 has been bound to 11.1.
```

To illustrate the whole refinement process, we present in Table 2 the ground proof opposite to the abstract proof: the left column contains the facts of the abstract proof and the right column contains the facts of the ground proof. The *mathematical* font has been used instead of `teletype` for typographical reasons. The symbol $\vdash$ stands for `PROVABLE` and $K$ for `k(all(x,diag(x)))`. The steps added during refinement can be grouped in two different categories. The first category (braced by a curly bracket) is composed of those steps having the first and the last line abstracted to the same formula. These steps corresponds to a single step of the abstract proof. For instance, line 2 of the abstract proof encodes seven steps of the ground proof, that is, the derivation of $\vdash \sim prov(all(x, diag(x)), K)$ from $\vdash\ all(x, diag(x))$ Proof lines 12, 13, and 14 are abstracted to the same abstract formula, $\vdash\ d$. The second category (braced by a square bracket) is composed by the steps needed to bridge gaps between ground inference rules. For instance, the single inference that allows the derivation of proof line 12 from proof lines 11 and 10 in the abstract proof (an application of the `taut` decider) is not valid in the ground space. We need to add six steps in the ground proof.

Having refined the outline, we must finally check that it is indeed a proof. That is, that every try is derived from previous tries. To do this we use the command `ol2prf` which attempts to convert an outline into a proof. This command checks that the outline is a well formed proof, translates the (pairs of) numbers `m.n` used to refer to tries to the natural numbers used to refer to lines of a proof, and asserts each try as a proof line.

```
metamaths:: show outline;
...
16.0  (not PROVABLE(all(x, diag(x)))) and (not PROVABLE(~ all(x, diag(x))))
      ANDI  7.0  15.0
16.1  exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
      EXISTI  16.0
17.0  exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
      MATCH  16.1
18.0  OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
      IMPI  OCONS  17.0
```

| | Abstract Proof | | | Ground Proof | |
|---|---|---|---|---|---|
| 1 | $\mathit{OCONS}$ | (1) | 11 | $\mathit{OCONS}$ | (11) |
| | | | 12 | $\vdash all(x, diag(x))$ | (12) |
| | | | 3 | $\vdash all(x, diag(x)) \wedge \mathit{SUBST}(\ldots) \supset\vdash diag(K)$ | |
| | | | 13 | $\vdash diag(K)$ | (10) |
| 2 | $\vdash d$ | (2) | 1 | $\vdash all(x, diag(x)\, equiv \sim prov(all(x, diag(x)), x)) \wedge$ $\wedge\, \mathit{SUBST}(\ldots) \supset\vdash diag(K)\, equiv \sim prov(all(x, diag(x)), K)$ | |
| | | | 2 | $\vdash diag(K)\, equiv \sim prov(all(x, diag(x)), K)$ | |
| | | | 4 | $\vdash diag(K) \wedge \vdash diag(K)\, equiv \sim prov(all(x, diag(x)), K) \supset$ $\vdash\sim prov(all(x, diag(x)), K)$ | |
| | | | 14 | $\vdash\sim prov(all(x, diag(x)), K)$ | (12) |
| 3 | $\vdash d \supset\vdash\sim d \wedge \mathit{NUMB}(k(d))$ | | 15 | $\vdash all(x, diag(x)) \supset\vdash prov(all(x, diag(x)), K) \wedge \mathit{NUMB}(K)$ | |
| 4 | $\vdash\sim d$ | (2) | 16 | $\vdash prov(all(x, diag(x)), K)$ | (10) |
| | | | 9 | $\mathit{CONS} \supset \neg\vdash prov(all(x, diag(x)), K) \vee \neg$ $\vdash\sim prov(all(x, diag(x)), K)$ | |
| 5 | $\mathit{CONS} \supset (\neg\vdash d \vee$ $\vee\neg\vdash\sim d)$ | | 10 | $\mathit{CONS} \supset \neg\vdash\sim prov(all(x, diag(x)), K) \vee$ $\vee\neg\vdash prov(all(x, diag(x)), K)$ | |
| 6 | $\mathit{FALSE}$ | (1 2) | 17 | $\mathit{FALSE}$ | (11 12) |
| 7 | $\neg\vdash d$ | (1) | 18 | $\neg\vdash all(x, diag(x))$ | (18) |
| 8 | $\vdash\sim d$ | (8) | 19 | $\vdash\sim all(x, diag(x))$ | (19) |
| 9 | $\exists n\, \neg(\mathit{NUMB}(n) \supset\vdash d)$ | (1 8) | 20 | $\exists n\, \neg(\mathit{NUMB}(n) \supset\vdash diag(n))$ | (11 19) |
| 10 | $\neg(\mathit{NUMB}(n) \supset\vdash d)$ | (10) | 21 | $\neg(\mathit{NUMB}(n) \supset\vdash diag(n))$ | (21) |
| | | | 5 | $\mathit{NUMB}(n) \supset\vdash diag(n) \vee \vdash\sim diag(n)$ | |
| | | | 22 | $\vdash\sim diag(n)$ | (21) |
| | | | 7 | $\vdash all(x, diag(x)\, equiv \sim prov(all(x, diag(x)), x)) \wedge \mathit{SUBST}(\ldots) \supset$ $\vdash diag(n)\, equiv \sim prov(all(x, diag(x)), n)$ | |
| | | | 8 | $\vdash diag(n)\, equiv \sim prov(all(x, diag(x)), n)$ | |
| | | | 6 | $\vdash\sim diag(n) \wedge \vdash diag(n)\, equiv \sim prov(all(x, diag(x)), n) \supset$ $\vdash prov(all(x, diag(x)), n)$ | |
| | | | 23 | $\vdash prov(all(x, diag(x)), n)$ | (21) |
| 11 | $(\mathit{NUMB}(n) \wedge \vdash\sim d) \supset\vdash d$ | | 24 | $\mathit{NUMB}(n) \wedge \vdash prov(all(x, diag(x)), n) \supset\vdash all(x, diag(x))$ | |
| 12 | $\vdash d$ | (1 8) | 25 | $\vdash all(x, diag(x))$ | (11 19) |
| 13 | $\mathit{CONS} \supset (\neg\vdash d \vee \neg\vdash\sim d)$ | | 25 | $\mathit{CONS} \supset (\neg\vdash all(x, diag(x)) \vee \neg\vdash\sim all(x, diag(x)))$ | |
| 14 | $\mathit{FALSE}$ | (1 8) | 26 | $\mathit{FALSE}$ | (9 18) |
| 15 | $\neg\vdash\sim d$ | (1) | 28 | $\neg\vdash\sim all(x, diag(x))$ | (11) |
| 16 | $\neg\vdash d \wedge \neg\vdash\sim d$ | (1) | 29 | $\neg\vdash all(x, diag(x)) \wedge \neg\vdash\sim all(x, diag(x))$ | (11) |
| 17 | $\exists w1\, (\neg\vdash w1 \wedge \neg\vdash\sim w1)$ | (1) | 30 | $\exists w1\, (\neg\vdash w1 \wedge \neg\vdash\sim w1)$ | (11) |
| 18 | $\mathit{OCONS} \supset \exists w1\, (\neg\vdash w1 \wedge \neg\vdash\sim w1)$ | | 31 | $\mathit{OCONS} \supset \exists w1\, (\neg\vdash w1 \wedge \neg\vdash\sim w1)$ | |

Table 2: Abstract and Ground proof.

```
metamaths:: ol2prf;
...
29   (not PROVABLE(all(x, diag(x)))) and
     (not PROVABLE(~ all(x, diag(x))))     (11)
30   exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (11)
31   OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
Congratulations: it's a proof!
```

This completes the construction of the ground proof from the outline provided by the abstract proof.

As a conclusive remark, it is important to notice that a lot of the process described in this section can be automated. As shown above in this section, the mapping back is completely automated (by the command `mapback proof`), as it is the process which takes in input an outline, checks whether it is a proof, and produces the corresponding proof (command `ol2prf`). The process which starts from the outline produced by `mapback` and outputs the outline input to `ol2prf`, instead, cannot be completely automated, not even in principle. In the general case, this process involves instantiating parameters and performing first order theorem proving between any two tries which must be connected. The advantage is that this should be easier than constructing the complete proof from scratch (as we would have to do if we did not use abstraction).

# 8 Other Incompleteness Proofs

Shankar [Sha86] has obtained a complete mechanical verification of Gödel's First Incompleteness Theorem using the Boyer and Moore theorem prover [BM79]. To construct this proof, he developed a (finite) set theory `Z2` and wrote a LISP-like interpreter as a formula in `Z2`. Using this interpreter, he was able to encode a function representing the provability of the theory within the theory. This gave him enough self-reference to perform the difficult construction of a Gödel sentence, `G` which asserts its unprovability. He then showed that if `G` is either provable or unprovable, then it is both provable and unprovable. But if `Z2` is complete, either `G` is provable or unprovable since it does not contains any free variables. Thus we cannot have both completeness and consistency for `Z2`. Shankar's proof is a very impressive and substantial piece of research. The events file to generate the proof is more than 1 Mbyte. Our proof is much more modest by comparison. We assume much which Shankar develops from first principles (*eg.* Gödel numbering, the existence of an encoding of provability within the theory, the construction of the Gödel sentence). Nevertheless, our proof can be seen as an abstraction of a more complete proof like Shankar's in which we only perform the top-level reasoning. To obtain a complete proof would require yet more refinement; it would be a very interesting exercise to try to perform this refinement within the `GETFOL` system. However, it is beyond the goals of this paper since our main aim is to demonstrate the usefulness of abstraction in helping us find a proof.

Quaife [Qua88] has also obtained a proof of Gödel's First Incompleteness Theorem using the ITP system. This proof is similar to the one given here since it assumes the diagonal lemma and only provides the high-level steps. Indeed, it is perhaps even simpler than our proof since Quaife encodes provability as the modal operator of $KT$. Quaife's proof therefore avoids the need to discuss several issues like substitution and universal quantification.

# 9 Conclusions

We have described an experiment in which a simplified proof of Gödel's First Incompleteness theorem was constructed with the aid of abstraction in the GETFOL proof checking system. To perform this experiment, a theoretical and practical framework for using abstraction in theorem proving was developed. We believe that this experiment convincingly demonstrates that the use of abstraction proposed in this paper can aid both the construction and explanation of proofs of difficult and challenging theorems.

# References

[AG93] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[BM79] R.S. Boyer and J.S. Moore. *A Computational Logic.* Academic Press, 1979. ACM monograph series.

[GAP95] E. Giunchiglia, A. Armando, and P. Pecchiari. Structured proof procedures. *Annals of Mathematics and Artificial Intelligence*, 15(I), 1995.

[Giu94] F. Giunchiglia. The GETFOL Manual - GETFOL version 2. Technical Report 94-0010, DIST - University of Genova, Genoa, Italy, 1994.

[Göd86] Kurt Gödel. Some metamathematical results on completeness and consistency (1930b). In *Kurt Gödel: Collected Works.* Oxford University Press, 1986.

[GW89] F. Giunchiglia and T. Walsh. Theorem Proving with Definitions. In *Proc. of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 175–183, 1989. Also IRST-Technical Report 8901-03 and DAI Research Paper No 429, University of Edinburgh.

[GW91] F. Giunchiglia and T. Walsh. Using abstraction. In *Proc. of the 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, Leeds, UK, 1991. Also IRST-Technical Report 9010-08 and DAI Research Paper 515, University of Edinburgh.

[GW92a] F. Giunchiglia and T. Walsh. A Theory of Abstraction. *Artificial Intelligence*, 57(2-3):323–390, 1992. Also IRST-Technical Report 9001-14, IRST, Trento, Italy.

[GW92b] F. Giunchiglia and T. Walsh. Tree subsumption: Reasoning with outlines. In *Proc. 10th European Conference on Artificial Intelligence ECAI-92*, pages 77–81, Vienna, Austria, 1992. Also IRST-Technical Report 9205-01, IRST, Trento, Italy.

[Kle52] S.C. Kleene. *Introduction to Metamathematics.* North Holland, 1952.

[Kno94] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, (68):243–302, 1994.

[Kor87] R.E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[McC90] W. W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Maths and CS. Division, Argonne National Laboratory, Argonne, Illinois, 1990.

[Pla81] D.A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[Pra65] D. Prawitz. *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[Qua88] A. Quaife. Automated proofs of Löb's Theorem and Gödel's two incompleteness theorems. *Journal of Automated Reasoning*, (4):219,231, 1988.

[Sha86] N. Shankar. *Proof Checking Metamathematics.* Phd thesis, University of Texas at Austin, 1986.

[Smo77] C. Smorynski. The Incompleteness Theorems. In Jon Barwise, editor, *Handbook of Mathematical Logic*, pages 821–865. North Holland Publishing Company, 1977.

[SVG94] R. Sebastiani, A. Villafiorita, and F. Giunchiglia. Proving Theorems by Using Abstraction Interactively. In *To appear in the proceedings of the Second International Round-Table on Abstract Intelligent Agent*, Rome, Italy, 1994. Also IRST-Technical Report 9403-17, IRST, Trento, Italy.

[VS94] A. Villafiorita and R. Sebastiani. Proof planning by abstraction. In *Proocedings of ECAI-94, Workshop: From Theorem Provers to Mathematical Assistants: Issues and Possible Solutions*, pages 15–24, 1994. MRG-DIST Technical Report 94-0025, DIST - University of Genova, Genova, Italy.

[Wey80] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13(1):133–176, 1980.

[WR25] A.N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, Cambridge, England, 1925.

# Appendix

## Ground Language

Below, in this subsection on the ground language, is the set of GETFOL commands which generate the language of metamaths. We have dropped the GETFOL prompt for reasons of space.

```
                                      comment ! PREDICATIVE SYMBOLS of arity ...        !
declare predconst PROVABLE 1;         comment ! ... 1                                   !
declare predconst NUMBER 1;           comment ! ... 1                                   !
declare predconst SUBST 4;            comment ! ... 4                                   !
                                      comment ! SENTENTIAL SYMBOLS                      !
declare sentconst CONS OCONS;         comment !                                         !
```

```
                                          comment ! FUNCTIONAL SYMBOLS of arity                !
declare funconst  k 1;                    comment ! ... 1 (prefix: standard binding priority) !
declare funconst  diag 1;                 comment ! ... 1 (prefix: standard binding priority) !
declare funconst  ~ 1     [pre=200];      comment ! ... 1 (prefix: binding priority is 200)   !
declare funconst  equiv 2 [inf=100 100];  comment ! ... 1 (infix: binding priority is 100)    !
declare funconst  all 2;                  comment ! ... 2 (prefix: standard binding priority) !
declare funconst  prov 2;                 comment ! ... 2 (prefix: standard binding priority) !
                                          comment ! INDIVIDUAL VARIABLES                       !
declare indvar   w1 w2;                   comment !                                            !
declare indvar   v1 t1;                   comment !                                            !
declare indvar   n;                       comment !                                            !
                                          comment ! INDIVIDUAL CONSTANTS                       !
declare indconst  x;                      comment !                                            !
```

## Ground Axiomatisation

```
 metamaths:: theory metamath:
  cons:  forall w1.(CONS imp not PROVABLE(w1) or not PROVABLE(~ w1))
  occ:   OCONS imp CONS
  oc:    OCONS imp not (forall n. (NUMBER(n) imp PROVABLE(diag(n))) and
         PROVABLE(~ all(x, diag(x))))

  corr:  forall w1.   (PROVABLE(w1) imp PROVABLE(prov(w1, k(w1))) and
         NUMBER(k(w1)))
  comp:  forall w2 n. (NUMBER(n) and PROVABLE(prov(w2, n)) imp PROVABLE(w2))

  numwise:   forall n. (NUMBER(n) imp PROVABLE(diag(n)) or PROVABLE(~ diag(n)))
  diagional: PROVABLE(all(x, diag(x) equiv ~ prov(all(x, diag(x)), x)))

  iffel:  forall w1 w2. (PROVABLE(w1) and PROVABLE(w1 equiv w2) imp
          PROVABLE(w2))
  iffers: forall w1 w2. (PROVABLE(~ w1) and PROVABLE(w1 equiv ~ w2) imp
          PROVABLE(w2))
  alle:   forall v1 w1 t1 w2. (PROVABLE(all(v1, w1)) and SUBST(t1, v1, w1, w2)
          imp PROVABLE(w2))

  triv1: SUBST(k(all(x, diag(x))), x, diag(x), diag(k(all(x, diag(x)))))
  triv2: SUBST(k(all(x, diag(x))), x, diag(x) equiv ~ prov(all(x, diag(x)), x)
         diag(k(all(x, diag(x)))) equiv
              ~ prov(all(x, diag(x)), k(all(x, diag(x)))))
  triv3: SUBST(n, x, diag(x) equiv (~ prov(all(x, diag(x)), x)),
               diag(n) equiv (~ prov(all(x, diag(x)), n)))
;
```

The GETFOL command theory allows us to define a GETFOL theory, that is a set of axioms, each with its own name, which can be globally invoked by certain commands, e.g. the command abstract theory (see below), by using the name of the theory, in this case metamath.

## Abstract Axiomatisation

```
 metamaths::   abstract theory metamath  by goedel;
 I am switching from the current context to: absmetamaths

 metamath

 cons : forall w1. (CONS imp ((not PROVABLE(w1)) or (not PROVABLE(~ w1))))
 occ : OCONS imp CONS
 oc : OCONS imp (not (forall n. (NUMBER(n) imp PROVABLE(d)) and PROVABLE(~ d)))
 corr : forall w1. (PROVABLE(w1) imp (PROVABLE(~ d) and NUMBER(k(w1))))
 comp : forall w2 n. ((NUMBER(n) and PROVABLE(~ d)) imp PROVABLE(w2))
```

```
numwise : forall n. (NUMBER(n) imp (PROVABLE(d) or PROVABLE(~ d)))
diagonal : PROVABLE(d equiv d)

iffel : forall w1 w2. ((PROVABLE(w1) and PROVABLE(w1 equiv w2)) imp PROVABLE(w2))
iffers : forall w1 w2. ((PROVABLE(~ w1) and PROVABLE(w1 equiv (~ w2))) imp PROVABLE(w2))
alle : forall v1 w1 t1 w2. ((PROVABLE(all(v1, w1)) and SUBST(t1, v1, w1, w2)) imp PROVABLE(w2))

triv1 : SUBST(k(d), x, d, d)
triv2 : SUBST(k(d), x, d equiv d, d equiv d)
triv3 : SUBST(n, x, d equiv d, d equiv d)

has been declared to be a new theory in 'absmetamaths'.
```

# Abstract Proof

```
absmetamaths:: show proof;
 1   OCONS     (1)
 2   PROVABLE(d)     (2)
 3   PROVABLE(d) imp (PROVABLE(~ d) and NUMBER(k(d)))
 4   PROVABLE(~ d)     (2)
 5   CONS imp ((not PROVABLE(d)) or (not PROVABLE(~ d)))
 6   FALSE     (1 2)
 7   not PROVABLE(d)     (1)
 8   PROVABLE(~ d)     (8)
 9   exists n. (not (NUMBER(n) imp PROVABLE(d)))     (1 8)
10   not (NUMBER(n) imp PROVABLE(d))     (10)
11   (NUMBER(n) and PROVABLE(~ d)) imp PROVABLE(d)
12   PROVABLE(d)     (1 8)
13   CONS imp ((not PROVABLE(d)) or (not PROVABLE(~ d)))
14   FALSE     (1 8)
15   not PROVABLE(~ d)     (1)
16   (not PROVABLE(d)) and (not PROVABLE(~ d))     (1)
17   exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (1)
18   OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
```

# Ground Proof

```
metamaths:: show proof;
   1   (PROVABLE(all(x, diag(x) equiv (~ prov(all(x, diag(x)), x)))) and
       SUBST(k(all(x, diag(x))), x,
             diag(x) equiv (~ prov(all(x, diag(x)), x)),
             diag(k(all(x, diag(x)))) equiv (~ prov(all(x, diag(x)), k(all(x, diag(x)))))))
       imp PROVABLE(diag(k(all(x, diag(x)))) equiv (~ prov(all(x, diag(x)), k(all(x, diag(x))))))
   2   PROVABLE(diag(k(all(x, diag(x)))) equiv (~ prov(all(x, diag(x)), k(all(x, diag(x))))))
   3   (PROVABLE(all(x, diag(x))) and
       SUBST(k(all(x, diag(x))), x, diag(x), diag(k(all(x, diag(x))))))
       imp PROVABLE(diag(k(all(x, diag(x)))))
   4   (PROVABLE(diag(k(all(x, diag(x))))) and
       PROVABLE(diag(k(all(x, diag(x)))) equiv (~ prov(all(x, diag(x)), k(all(x, diag(x)))))))
       imp PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x)))))
   5   NUMBER(n) imp (PROVABLE(diag(n)) or PROVABLE(~ diag(n)))
   6   (PROVABLE(~ diag(n)) and
       PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n))))
       imp PROVABLE(prov(all(x, diag(x)), n))
   7   (PROVABLE(all(x, diag(x) equiv (~ prov(all(x, diag(x)), x)))) and
       SUBST(n, x,
             diag(x) equiv (~ prov(all(x, diag(x)), x)),
             diag(n) equiv (~ prov(all(x, diag(x)), n))))
       imp PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))
   8   PROVABLE(diag(n) equiv (~ prov(all(x, diag(x)), n)))
   9   CONS imp ((not PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))) or
                 (not PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x)))))))
  10   CONS imp ((not PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x)))))) or
```

```
             (not PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))))
11    OCONS     (11)
12    PROVABLE(all(x, diag(x)))     (12)
13    PROVABLE(diag(k(all(x, diag(x)))))     (12)
14    PROVABLE(~ prov(all(x, diag(x)), k(all(x, diag(x)))))     (12)
15    PROVABLE(all(x, diag(x))) imp (PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))
      and NUMBER(k(all(x, diag(x)))))
16    PROVABLE(prov(all(x, diag(x)), k(all(x, diag(x)))))     (12)
17    FALSE     (11 12)
18    not PROVABLE(all(x, diag(x)))     (11)
19    PROVABLE(~ all(x, diag(x)))     (19)
20    exists n. (not (NUMBER(n) imp PROVABLE(diag(n))))     (11 19)
21    not (NUMBER(n) imp PROVABLE(diag(n)))     (21)
22    PROVABLE(~ diag(n))     (21)
23    PROVABLE(prov(all(x, diag(x)), n))     (21)
24    (NUMBER(n) and PROVABLE(prov(all(x, diag(x)), n))) imp PROVABLE(all(x, diag(x)))
25    PROVABLE(all(x, diag(x)))     (11 19)
26    CONS imp ((not PROVABLE(all(x, diag(x)))) or (not PROVABLE(~ all(x, diag(x)))))
27    FALSE     (11 19)
28    not PROVABLE(~ all(x, diag(x)))     (11)
29    (not PROVABLE(all(x, diag(x)))) and (not PROVABLE(~ all(x, diag(x))))     (11)
30    exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))     (11)
31    OCONS imp exists w1. ((not PROVABLE(w1)) and (not PROVABLE(~ w1)))
```